Aspects of sustainable software design for complex robot platforms in multi-disciplinary research projects on embodied cognition

Hülse, Martin; Lee, Mark

Publication date:
2008

Citation for published version (APA):
Hülse, M., & Lee, M. (2008). *Aspects of sustainable software design for complex robot platforms in multi-disciplinary research projects on embodied cognition*. Abstract from Intelligent Robots and Systems, Nice, France. http://hdl.handle.net/2160/1868

Aspects of sustainable software design for complex robot platforms in multi-disciplinary research projects on embodied cognition

Martin Hülse and Mark Lee

***Abstract*— Sophisticated robot systems have become an important part in cognition research. On the one hand, autonomous robots are intended to provide a proof of concept for cognitive models. On the other hand, cognition research becomes a source of inspiration in targeting current limitations in the engineering of robust, flexible and adaptive artifacts. In this work, we discuss aspects of software development and integration for heterogeneous robotic systems in cognition research. As we will argue, one important issue is the combination of different computational paradigms within one robot system, which are rooted in the divergent approaches of engineers and scientists. This discussion lead to the introduction of a software framework aiming to overcome some well known problems of sustainable software development in robotics, but particular important for multi-disciplinary and multi-center cognition research projects. The introduced framework is based on well established standards in software engineering and therefore can be considered for a wide range of cognition research platforms and projects. Further on, we will briefly present a robotic setup where this framework is applied. It consists of a manipulator of 14 DOF (degrees of freedom) and an active vision system of 4 DOF. It is part of research activities aiming to model behavior integration and action-selection mechanisms based in large-scale neural networks.**

I. INTRODUCTION

The progress in robotic manipulation and mobile robots makes nowadays an autonomous robot platform more than an object of investigation for its own right. Miniaturization has led to platforms equipped with high dimensional sensors and actuators with many degrees of freedom able to enter the daily environment of human beings. In consequence, the focus of research and development is turning to robust, multi-modal, multi-functional and adaptive interaction of an autonomous robot system in a complex and dynamic environment. The creation of artefacts of such flexibility is still a challenge, especially if scalability is considered.

Cognition research has become one source of inspiration as well as a guidance to overcome current limitations in engineering of more complex and adaptive systems. On the other hand, cognition research projects have been utilizing robot systems as demonstrators and therefore they serve as an important proof of concept in this field. Furthermore, embodied cognition, in particular, is focused on the crucial role the body has for the development of cognitive behavior and therefore it becomes rather usual that experiments in this research involve robot systems of arbitrary complexity.

As soon as sophisticated robotic systems become part of a cognition research project one is facing a multi-disciplinary and usually also a multi-center project. Robotic engineers and scientists from different fields have to combine their approaches and know-how in order to create systems beyond the current state-of-the-art. One important area of this combination is software development. Usually multi-disciplinary research projects cannot start from scratch building up a new system. They are rather confronted with a task of integration, in which specific and very different software components are combined to one complex system. Very often these software components have been developed over years and are tightly bonded to specific constraints, such as operation system, programming language, middleware. The crucial point for robotic platforms in cognition research projects is the difference between the domain of cognition research and engineering, which is also indicated by the difference of the applied software frameworks [1]. Due to missing standards in robotics it is already difficult to extend or integrate robot systems without considering high-level cognitive models. Hence the integration of software developed in the domain of cognitive science and robotics becomes rather a challenge of its own.

The objective of this paper is to outline crucial aspects that become relevant in software engineering of robotic systems in cognition research projects. Based on our experience and recent reviews on software development and integration in robotics we have developed a framework for medium and large projects aiming for complex experimental robotic platforms for cognitive models. This framework is purely conceptual, based on design patterns and standards in software engineering, and can therefore be applied to any hardware and software environment. Furthermore, we will explain some elements of this framework in detail, based on examples of an ongoing project basically involving a manipulator equipped with a three-finger system and a stereo-vision system. This experimental platform is used for the development of large-scale neural models coordinating reaching and grasping tasks.

This paper is organized as follows. The next section introduces the key aspect influencing the software integration in cognition research projects. After this, the following two sections give an overview of the-state-of-the-art in software development / integration in robotics and outline our proposal of a framework considering aspect of robotics in cognition research. This is followed by a section which gives a concrete example of this framework leading to the concluding section of this work.

Dept. Computer Science, Aberystwyth University, Penglais, SY23 3DB, Wales, UK {msh,mhl}@aber.ac.uk

II. KEY ISSUES OF SOFTWARE DEVELOPMENT IN ROBOTICS

The problem of sustainable software design has been an important issues in software engineering in general and therefore is widely and continuously discussed in many fields and domains. Guiding issues like modular, interoperable and reusable software have led to the promotion of object-oriented design principles, design patterns as well as agile and test-driven software design methods. These issues are relevant in robotics indeed, but shall not be discussed here. Nevertheless, the following collection outlines specific aspects of software design crucial for robotic related multi-disciplinary research projects.

A. Prototypes and multi-components

Sophisticated robot systems are build up of different components. Sensors, actuators and mechanics supposed to establish a coherent robot system are very often products of different manufactures. Sometimes these components have even the character of a prototype, i.e. software and hardware are not sufficiently tested and might lack in specific functionalities and robustness. Furthermore, it is not unusual that the delivered driver software of hardware devices is very rudimentary, though one might expect software providing already solved and well known standard applications.

In consequence, for robotic components the first step is the development of a software which provides robust and general functionality and a proper error handling mechanism. This includes also sufficient test cases, which support a robust and smooth exchange of system components, if firmware and / or hardware devices must be upgraded or exchanged.

B. Different representations and levels of abstraction

In research laboratories it is common that one device, i.e. a whole robot system or a specific component, is used for experiments in different domains. This might be necessary because experiments in a project must be conducted on a lower level of functionality in oder to decide future design issues. On the other hand, a research laboratory might be involved in other projects, currently or in future, and so it is essential that specific components can efficiently be used in very different experiments.

Therefore it becomes important for the software design to provided different levels of abstraction and data representations and of course this should take as little effort as possible. This refers to the need that the core functionality of robotic devices can be used independently, and that the exchange and extension of system services with respect to hardware and software must be provided.

C. Distribution of computational resources

Robotic system components might only work in a specific software environment. Some devices might also run on specific hardware, such as FPGAs. It is also usual that computational expensive processes have to be distributed over different computers in order to guarantee real-time constraints. Recent experiments in neuroscience also show that clusters might be necessary to simulate large-scale neural models driving a robot platform [2]. Hence, nowadays robust, transparent and reliable interprocess communication is a need for almost any nontrivial autonomous robot system.

D. Distributed teams

Where cognitive science and robotics meet it is very likely that developers of specific system components are geographically distributed. The exchange of source codes and software libraries (sometimes even only pre-compiled) via suitable software repositories becomes only one major issue to consider in this process. Due to the division of knowledge and competence within a project it also very likely that software integration between the different partners is rather vertical instead of horizontal.

Horizontal integration means that two or more project partners deliver software which is horizontally organized within the overall software architecture. For instance, one team delivers the hardware and software of sensor type *A*, while another team is doing so for sensor type *B* and another team is responsible for an actuator *C*. All three software components can be developed independently.

A vertical integration starts if a fourth party develops applications *X* on top *A*, *B* and *C*, taking data from *A* and *B* and generating data feed into *C*. The success of this type of integration depends on very carefully defined and implemented interfaces. Since formal interface definition languages are purely syntactic and cannot cover any semantic information, this process must involve an understanding of the constraints and needs of each part in a reasonable depth. This usually requires time, rather days than hours.

E. Simulator

Almost every complex robotic project sooner or later requires the use of a simulator, especially if an autonomous robot system is intended as a test platform for learning or other forms of self-organized mechanism of adaptation. Simulations are an important tool to provide a proof of concept for new methods and in order to tune important system parameters in advance. However, it only makes sense to use simulators if the control software under investigation generates the same qualitative behavior in simulation as on the real robot. It is also important that the same control software can directly be used for both, simulator and real platform, without any parameter changes or even refactoring.

F. Integration of different paradigms

Robotic related cognition research projects have to pay particular attention to the coupling between high-level cognitive models and hardware specific software. Cognitive models are grounded in specific paradigms of computation and knowledge representation. Consequently, this leads to model-implementations based on declarative or functional computer languages or even simulations of neural networks. In contrast hardware-close software is usually developed in procedural computer languages strictly following this paradigm.

The problem with different paradigms is that sometimes specific constraints present either in the higher-level model or

in the lower-level software cannot directly be represented in the other domain. Hence, these constraints cannot be handled at all and therefore cut the overall system performance. Examples relevant in almost any systems combining robot hardware and cognitive models are real-time constraints and the different time-scales that specific system components are operating on.

In consequence a lot of effort must be put into developing efficient pre- and post-processing, scheduling and error-handling for bridging robot hardware and cognitive models.

G. Flexibility

The aspect of different paradigms leads to another problem, also described in [1]. An engineer creates systems, whose component functions are most efficient when they meet a detailed set of specifications exactly. The consequence is high performance for a very specific task. But as soon as the application domain is extended or becomes more general a decline of performance must be expected.

On the other hand, higher-level robotic applications, and cognitive scientists are no exception, they develop their models, applications and experiments in a language grounded in an ontology based on general principles. Hence, they expect reasonable and scalable performance for general domains and problem spaces.

The aspect of interface definition and description was already described in section II-D for geographically distributed teams. In this context, interface definition and implementation become crucial because of the multi-disciplinary character of cognitive science and engineering, thus, the different approaches and the divergent expectations of specific and general system performance.

In fact, one has to accept the inevitability of different understandings between cognitive scientists and engineers about the needs and the relevance of specific elements of the targeted models and tasks. This discrepancy is often overseen at the project-start but will emerge as soon as lower and higher level implementations meet. As a matter of fact, the re-definition of interfaces, frameworks or even experiments will be the consequence. In our experience such re-definitions will happen several times in larger projects and always go hand in hand with refactoring of certain extents. It is therefore, important to be aware of this problem, and on the other hand to provide a software engineering framework which allows, with reasonable effort, the alteration of the interfaces and the corresponding implementations on both sides: robotic hardware functionality and high-level cognitive models.

III. STATE OF THE ART

Robotics community is aware of the first five problems issued in II-A – II-E, but very little attention is focused on the problem of different paradigms. Nevertheless, standards providing robust and flexible solutions for interoperable, reusable robotic software does not exist yet. Although this lack of standards is recognized by many researchers, the most common solution to overcome this problem is to develop a new software. Mostly such implementations are claimed to be more general, but indeed, are addressing only specific needs and even more crucial the software is even immature. Consequently, it is not used by other labs and therefore is far away from providing a base for any standard.

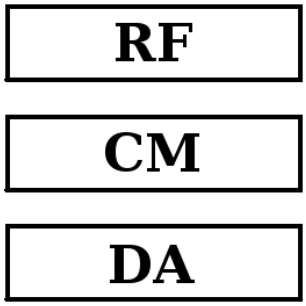

Fig. 1. The three software architecture layers of system design in robotics. *DA*, driver and algorithms, *CM* interprocess communication layer and *RF* robotic control framework, see also [3], [4] and [10].

Noticeable is the effort in many robotic projects developing middleware for a framework of handling distributed robotic systems. However, interprocess communication is an important aspect but not the only one for autonomous robots. Recent reviews [3], [4] show that in robot systems basically a 3-layered software architecture must be considered (see Fig. 1). In the lower level *DA* software driver and algorithm implementations are located. The middle level *CM* provides interprocess communication. The top level *RF* is the place where the actual robotic control frameworks are developed. It is this top layer where robotic projects implement their strategies and models, generating intelligent behavior. Some research projects claim that this level should provide a declarative programming framework [3], because they think it is the best way to implement intelligent behavior. However, other projects would probably disagree introducing a different framework for this level, which matches best with their paradigm of computational intelligence.

Being aware of the subjectivity and biased view on the top level, current activities in developing general robotic programming frameworks are primarily focused on the two lower levels: *DA* and *CM*. *Player* [5], for instance, delivers a framework, where *DA* and *CM* are interwoven [4]. The *YARP* software [6] actually provides only a framework for the *CM* layer. The developers of *MIRO* [7] had similar intentions. However, they have built *MIRO* as an extension of *CORBA* in order to make this powerful middleware standard easier to handle and faster to learn. *ROCI* [8] is based on the philosophy that complex robot behavior is achieved by "wiring" irreducible modules. In consequence, this software provides the design of modules acting in a decentralized manner. Therefore, in *ROCI* all three layers collapse into one network of interacting primitive modules.

Another strategy called *MARIE* [9] tries to support the reuse of existing programming environments and their connections through a common middleware framework. Being aware about the missing standards in interprocess commu-

nication *MARIE* provides basically a set of design patterns able to integrate communication protocols present in systems composed of heterogeneous hardware components. However, the whole design is based on middleware.

Once again, these examples represent the focus on the *CM* level, which shows that system design is almost completely seen as a problem of reliable communication between the components. However, reliable and transparent communication has always an offset. This offset is crucial if many relative primitive interacting components have to cope with real-time constraints. What we want to emphasize is, if a framework based on interprocess communication is applied for system integration, it follows that, the lower the level of system functions the more the reduction of system performance due to the offset of communication.

It is this observation, that led us to the formulation of a framework which tries to keep the middle layer *CM* as "thin and high level" as possible. However, interprocess communication provided in *CM* is an essential part in order to connect high-level cognitive models and robotic hardware. But in using it very sparely one can apply computational expensive but standardized and mature middleware solutions. In doing so, one has a wide coverage of different software environments and on the same time one can handle many effective real-time constraints on lower level functions without the involvement of computationally expensive middleware. This reduces also the effort needed for refactoring the interfaces between high-level cognitive models and robot hardware. However, the problem of modular, interoperable and reusable software design in the basic layer *DA* must still be addressed explicitly. We have done this by the usage of specific design patterns, which will be explained in the next section.

IV. GENERAL FRAMEWORK

The general framework of our software architecture is based on the the three-layered architecture as show in Fig. 1. However, in the need for the support of a sustainable software design in *DA* we have divided this lower level into two levels: *API* and *MVC* (see Fig. 2).

The lower *API* provides simple and almost purely hardware related application interfaces. These interfaces provide common and general functionality for specific hardware devices, such as cameras, laser scanners, actuators. Although these implementations will be usually very simple and straight forward, they shall already make use of an object-oriented design. Also important is the testability of each component and the support for other software developers through documentations and basic example applications. It is also necessary that the components in the *API* layer can independently be used and developed. This ensures the smooth integration or update of new hardware and firmware.

On top of these APIs we only develop new system functions based on the model-view-control design pattern [11]. This design pattern supports a complete separation of hardware from applications and the first level of abstraction. While the model element provides all the hardware functionality the view and control processes can independently

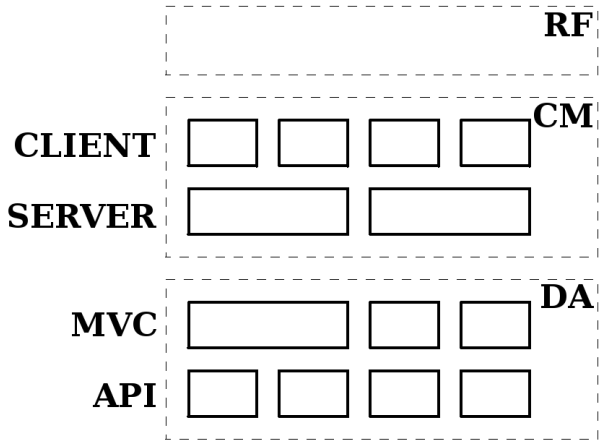

Fig. 2. The proposed software architecture framework in relation to the 3-layered variation in Fig. 1.

and parallel interacting with model. Different views can provide different representations of the current data even taking into account temporal aspects. Control elements can either monitor and maintain defined constraints or instantiate more sophisticated control schemas. It is also possible to combine several *API* components within one model.

As we have argued above, the middle level *CM* primarily connects the cognitive model implementations in *RF* with the robot hardware and related services. Since the software environment for cognitive models and robotic hardware is probably very different, it is recommended to use standard middleware solutions in order to cover as much diversity as possible. And to our knowledge, these standards in the domain of distributed systems will all support client-server frameworks. Hence, robotic functionalities and services can be provided by one or more server applications, while clients are responsible to request and deliver data needed and generated by the cognitive model running in the *RF* layer. Notice, the usage of standard middleware solutions also provides the distribution of lower robot functions, because different servers can run on different machines.

V. ROBOTIC SETUP FOR THE REVERSE ENGINEERING OF THE VERTEBRATE BRAIN

The above introduced framework is applied in a project, called REVERB [14], in which behavior integration and action-selection mechanisms are modelled based on biologically inspired large-scale neural networks. These models are tested and developed on a robot platform basically consisting of a 14 DOF (degrees of freedom) manipulator and a vision system. The manipulator integrates a 7 DOF *Lightweight arm LWA3* and a 7 DOF *Dextrous Hand SDH*. Both devices are manufactured by SCHUNK GmbH & Co. KG [13]. The vision system is based on a 4 DOF pan-tilt-verge platform equipped with two firewire cameras and a *SCAMP* vision system [12]. The unique feature of the *SCAMP* system is basically the pixel-per processor vision chip based on analog technology. This allows the execution

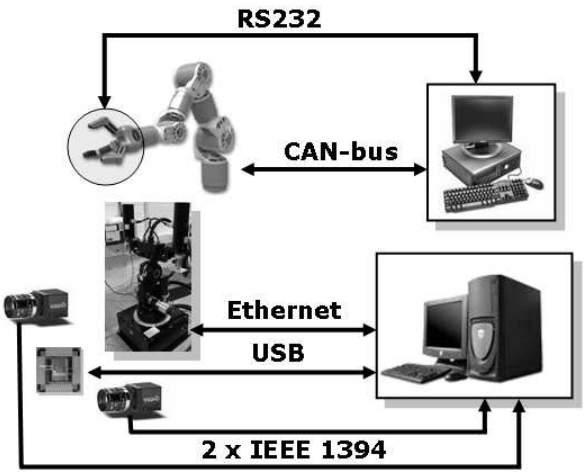

Fig. 3. Robot hardware and corresponding distribution over two PCs. The overall experimental platform consists of a manipulator of 14 DOF and vision system involving a pan-tilt-verge system, two firewire cameras and a SCAMP vision system [12].

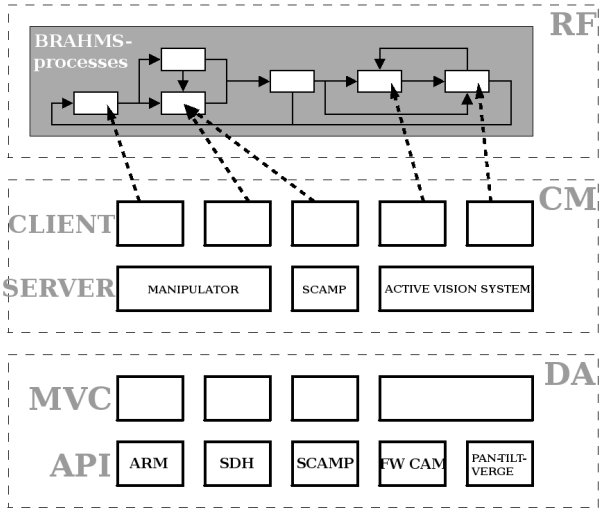

Fig. 4. Software architecture, see text for explanation.

of computational expensive image processing algorithms in real-time.

Almost typical for the integration of different devices, each is based on a different communication channel, such as CAN-bus, serial, USB, firewire and ethernet. Currently, the two main hardware components are even connected to different computers, (see Fig. 3).

A. Software architecture

Our software architecture has five independent components in the API layer; each for every hardware device: camera, pan-tilt-verge system, *LWA3*, *SDH* and *SCAMP*. On top the *MVC* layer integrate some but not all of these components. *LWA3*, *SDH* and *SCAMP* have still their own model-view-control implementation, while pan-tilt-verge and cameras are integrated in one pattern (Fig. 4). This does not mean that in the future no other additional patterns will summarize other components. This depends on the development in the project.

The applications in the *MVC* layer are wrapped by CORBA server implementations providing an interface for interprocess communication and distribution. CORBA-client implementations in arbitrary software environments are now able to access these hardware components and the services provided in the *MVC*-level. Due to usage of CORBA the interfaces must be written in IDL (interface description language). This provides, at least on the syntactical level, coherent interface definitions between low and high level functionality.

Actually CORBA-clients are part of the processes which establish the overall target of this software organization, that is the cognitive model implementation. As we have mentioned, the cognitive model in this process is implemented by large-scale artificial neural networks. The software used to simulate these networks is called *BRAHMS* [15]. Among many features, with *BRAHMS* one is able to link different processes in an arbitrary manner. In such a way, different layers of artificial neural networks can be connected, even recurrent. The processes can be implemented within a general C++ environment and the connectivity with other processes is defined in a XML-based language. It is also possible to simulate the system in a MATLAB environment.

Due to its general character, CORBA-clients can straightforward be instantiated in *BRAHMS*-process. In such a way, a distributed robot systems becomes part of a large-scale neural model, which is simulated in *BRAHMS* and might itself be executed on a cluster.

B. The usage of design patterns

The benefit of model-view-control based implementations might be best briefly demonstrated by the following two examples. We have outlined above that reusability of software involves the alterations of data representations and the level of abstractions. For visual information this means, that image data might be applied to different filters or feature detection processes. Hence, we have used the views in the MVC-design pattern in order to deliver different filters. The instantiations of the view processes operate independently and parallel. On one side, this supports the exploitation of multi-processor system, but more important, the implemented filters can be applied to any future instantiations of the corresponding design pattern. Therefore, a set of independently used functions can be generated which is totally separated from the underlying hardware.

As only one example for the *LWA3* 7 DOF arm system we have implemented a simple arm coordination task based on two independently working control processes. The arm coordination task is simply: while arm is moving, the orientation of the last segment, the hand segment, shall remain the same.

The corresponding MVC-pattern is initiated with only two control process. The first is responsible for the global orientation of the arm, while task for the second process is

to keep the orientation of the SDH hand in space constant. The bottom-line of this example is that the first process is actively changing the global configuration of the arm, while the second is passively adjusting the remaining DOF, which, in this case, maintains the orientation of the hand. Both control processes are operating in parallel on the same data. This avoids inconsistencies and makes the overall control much easier.

C. Switch between simulator and real robot system

The usage of CORBA supports also the integration of simulators. As we have seen it for the robot hardware a CORBA server can also be based on a robot simulator. If both server implementations are based on the same interface definition given in IDL, it makes actually no difference for a client to which server it is talking. Hence, without any changes in the client, it can communicate either with the simulator or with the real robot. This type of integration is successfully applied, for instance, for the mobile robot platform KURT3D and the corresponding simulation MACSim [16], [17].

D. Summary

The brief introduction of our robotic platform already outlines the importance of two key aspects in our software architecture: *MVC* design patterns and CORBA as widely supported middleware standard. The *MVC* patterns guarantee the strict separation of APIs and application layer right from the beginning. This supports the independent, modular, test-driven, and scalable software design of robotic components. We have also elucidated, how *MVC* patterns can simplify the control and provide different data representations. Complex, multi-modal and computationally expensive algorithms can already be implemented in the *MVC*-level without the usage of middleware.

The usage of *MVC* allows the integration of the *CM* layer on a much higher level of abstraction, which can lead to the reduction of interprocess communication. Therefore, powerful and computationally expensive middleware standards, like CORBA, can be applied without violating real-time constraints in the overall system. As we see in our example CORBA supports as wide range of software environments, which enables us to couple our robot hardware with a MATLAB framework. Further on, the IDL used in CORBA provides robust interface definitions between different developer teams and totally different data sources, such as a simulator. It is this last issue, which enables us to run a cognitive model either on a real robot or a simulator without any changes.

VI. CONCLUSION

Focused on current standards in software engineering we have introduced a software architecture particularly developed for robotic systems made of heterogeneous hardware devices and components. We have outlined how model-view-control design patterns and CORBA, as the leading middleware standard, can provide a sustainable software development for different levels of abstraction. As we have argued, this supports the integration of different computational paradigms. The last aspect makes our framework particularly interesting for robotics in cognition research, where engineers and scientists from different fields must integrate their different ways of system design and modelling.

VII. ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of EPSRC through grant EP/C516303/1.

REFERENCES

[1] A. Farinelli, G. Grisetti and L. Iocchi, ”Design and implementation of modular software for programming mobile robots” *Int. J. of Advanced Robotic Systems* vol. 3, 2006, pp. 37-42

[2] G.M. Edelman, ”Learning in and from Brain-Based Devices” *Science*, vol. 318(5853), November, 2007, pp. 1103-1105

[3] I.A.D. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I-H. S. and D. Apfelbaum. ”CLARAty: Challenges and Steps Toward Reusable Robotic Software” *Int. J. of Advanced Robotic Systems* vol. 3, 2006, pp. 23-30

[4] A. Makarenko, A. Brooks and T. Kaupp, ”On the Benefits of Making Robotic Software Frameworks Thin” *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'07)*, San Diego CA, USA, 2007.

[5] T.H.J. Collett, B.A. MacDonald and B.P. Gerkey, ”Player 2.0: Toward a Practical Robot Programming Framework” *Proc. of the Australasian Conf. on Robotics and Automation (ACRA 2005)*, Sydney, Australia, December 2005.

[6] G. Metta, P. Fitzpatrick and L. Natale, ”YARP: Yet Another Robot Platform” *Int. J. of Advanced Robotic Systems* vol. 3, 2006, pp. 43-48.

[7] G.K. Kraetzschmar, H. Utz, S. Sablatnög, S. Enderle and G. Palm ”Miro - Middleware for Cooperative Robotics” *RoboCup 2001: Robot Soccer World Cup V*, LNCS 2377, Springer, pp. 411-416.

[8] A. Farinelli, G. Grisetti and L. Iocchi, ”Design and implementation of modular software for programming mobile robots” *Int. J. of Advanced Robotic Systems* vol. 3, 2006, pp. 37-42.

[9] C. Cote, D. Letourneau, F. Michaud and Y. Brosseau, ”Robotics System Integration Frameworks : MARIE's Approach to Software Development and Integration” Springer Tracts in Advanced Robotics : Software Engineering for Experimental Robotics, Springer, vol. 30, March 2007.

[10] R.P. Bonasso, D. Kortenkamp, D.P. Miller and M.G. Slack, ”Experiences with an Architecture for Intelligent Reactive Agents” *Proc. of the Int. Joint Conf. on Artificial Intelligence*, 1995.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlisside ”Design patterns: Elements of Reusable Object-Oriented Software” *Addidon-Weslay*, USA, 1995.

[12] D. R. W. Barr, P. Dudek, J. Chambers and K. Gurney, ”Implementation of Multi-layer Leaky Integrator Networks on a Cellular Processor Array” *Int. Joint Conf. on Neural Networks*, IJCNN 2007, USA, 2007.

[13] SCHUNK GmbH & Co. KG, *http://www.schunk.com*, 2007.

[14] REVERB: Reverse Engineering the VERtebrate Brain, *http://www.abrg.group.shef.ac.uk/projects/reverb/public/* EPSRC Research Grant no. EP/C516303/1, UK, 2007.

[15] BRAHMS, *http://sourceforge.net/projects/abrg-brahms*, 2007.

[16] Rome, E.; Paletta, L.; Sahin, E.; Dorffner, G.; Hertzberg, J.; Breithaupt, R.; Fritz, G.; Irran, J.; Kintzler, F.; Lörken, C.; May, S.; Ugur, E. The MACS project: An approach to affordance-inspired robot control. to appear in: Towards Affordance-based Robot Control, Proceedings of Dagstuhl Seminar 06231, Springer, LNAI 4760, Rome, E., Hertzberg, J. and Dorffner, G. (eds.), 2007

[17] E. Ugur, M.R. Dogar, M. Cakmak and E. Sahin, ”The learning and use of traversability affordance using range images on a mobile robot”, *Proc. of IEEE Int. Conf. on Robotics and Automation*, ICRA07, 2007.